6.1 Naive Gaussian Elimination

Our objective in this chapter is to develop a good program for solving a system of n linear equations in n unknowns:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n = b_3 \\ \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad\quad\quad \vdots \quad\quad \vdots \\ a_{i1}x_1 + a_{i2}x_2 + a_{i3}x_3 + \cdots + a_{in}x_n = b_i \\ \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad\quad\quad \vdots \quad\quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n \end{cases} \tag{1}$$

In compact form, this system can be written as

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \qquad (1 \leqq i \leqq n)$$

In these equations, a_{ij} and b_i are prescribed real numbers (data) and the unknowns x_j are to be determined. Subscripts on the letter a are separated by a comma only if necessary for clarity—for example, in $a_{32,75}$ but not in a_{ij}.

Numerical Example

In this section, the simplest form of Gaussian elimination is explained. The adjective *naive* applies because this form is not suitable for automatic computation unless essential modifications are made. We illustrate with a specific example that has four equations and four unknowns:

$$\begin{cases} 6x_1 - 2x_2 + 2x_3 + 4x_4 = 16 \\ 12x_1 - 8x_2 + 6x_3 + 10x_4 = 26 \\ 3x_1 - 13x_2 + 9x_3 + 3x_4 = -19 \\ -6x_1 + 4x_2 + x_3 - 18x_4 = -34 \end{cases} \tag{2}$$

In the first step of the elimination procedure, certain multiples of the first equation are subtracted from the second, third, and fourth equations so as to eliminate x_1 from these equations. Thus, we want to create 0's as coefficients for each x_1 below the first (where 12, 3, and -6 now stand). It is clear that we should subtract 2 times the first equation from the second. (This multiplier is simply the quotient $\frac{12}{6}$.) Likewise, we should subtract $\frac{1}{2}$ times the first equation from the third. (Again, this multiplier is just $\frac{3}{6}$.) Finally, we should subtract -1 times the first equation from the fourth. When all of this has been done, the result is

$$\begin{cases} 6x_1 - 2x_2 + 2x_3 + 4x_4 = 16 \\ -4x_2 + 2x_3 + 2x_4 = -6 \\ -12x_2 + 8x_3 + x_4 = -27 \\ 2x_2 + 3x_3 - 14x_4 = -18 \end{cases} \tag{3}$$

Note that the first equation was not altered in this process, although it was used to produce the 0 coefficients in the other equations. In this context, it is called the **pivot** equation.

Notice, also, that Systems (2) and (3) are *equivalent* in the following technical sense: Any solution of (2) is also a solution of (3), and vice versa. This follows at once from the fact that if equal quantities are added to equal quantities, the resulting quantities are equal. One can get System (2) from System (3) by adding 2 times the first equation to the second, and so on.

In the second step of the process, we mentally ignore the first equation and the first column of coefficients. This leaves a system of three equations with three unknowns. The same process is now repeated using the top equation in the smaller system as the current pivot equation. Thus, we begin by subtracting 3 times the second equation from the third. (The multiplier is just the quotient $\frac{-12}{-4}$.) Then we subtract $-\frac{1}{2}$ times the second equation from the fourth. After doing the arithmetic, we arrive at

$$\begin{cases} 6x_1 - 2x_2 + 2x_3 + 4x_4 = 16 \\ -4x_2 + 2x_3 + 2x_4 = -6 \\ 2x_3 - 5x_4 = -9 \\ 4x_3 - 13x_4 = -21 \end{cases} \tag{4}$$

The final step consists in subtracting 2 times the third equation from the fourth. The result is

$$\begin{cases} 6x_1 - 2x_2 + 2x_3 + 4x_4 = 16 \\ -4x_2 + 2x_3 + 2x_4 = -6 \\ 2x_3 - 5x_4 = -9 \\ -3x_4 = -3 \end{cases} \tag{5}$$

This system is said to be in **upper triangular** form. It is equivalent to System (2).

This completes the first phase (**forward elimination**) in the Gauss algorithm. The second phase (**back substitution**) is solving System (5) for the unknowns *starting at the bottom*. Thus, from the fourth equation,

$$x_4 = \frac{-3}{-3} = 1$$

Putting $x_4 = 1$ in the third equation gives us

$$2x_3 - 5 = -9$$

whence

$$x_3 = \frac{-4}{2} = -2$$

and so on. The solution is

$$x_1 = 3 \qquad x_2 = 1 \qquad x_3 = -2 \qquad x_4 = 1$$

This example can be solved directly using Maple as follows.

```
with(linalg):
A := array([[ 6,  -2, 2,   4],
            [12,  -8, 6,  10],
            [ 3, -13, 9,   3],
            [-6,   4, 1, -18]]);
b := array( [16, 26, -19, -34]);
linsolve(A, b);
```

Algorithm

To simplify the discussion, we write System (1) in matrix-vector form. The coefficient elements a_{ij} form an $n \times n$ square array, or matrix. The unknowns x_i and the right-hand-side elements b_i form $n \times 1$ arrays, or vectors.* Hence, we have

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{i1} & a_{i2} & a_{i3} & \cdots & a_{in} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix} \tag{6}$$

or

$$Ax = b$$

Operations between equations correspond to operations between rows in this notation. We shall use these two words interchangeably.

*To save space, we occasionally write a vector as $[x_1, x_2, \ldots, x_n]^T$, with the T standing for the transpose. It tells us that this is an $n \times 1$ array or vector and not $1 \times n$, as would be indicated without the transpose. (See Appendix A for linear algebra notation and concepts.)

Now let us organize the naive Gaussian elimination algorithm for the general system, which contains n equations and n unknowns. In the forward elimination phase of the process, there are $n - 1$ principal steps. The first of these steps uses the first equation to produce $n - 1$ zeros as coefficients for each x_1 in all but the first equation. This is done by subtracting appropriate multiples of the first equation from the others. In this process, we refer to the first equation as the first **pivot equation**. For each remaining equation ($2 \leqq i \leqq n$), we compute

$$\begin{cases} a_{ij} \leftarrow a_{ij} - \left(\dfrac{a_{i1}}{a_{11}}\right) a_{1j} & (1 \leqq j \leqq n) \\ b_i \leftarrow b_i - \left(\dfrac{a_{i1}}{a_{11}}\right) b_1 \end{cases}$$

The symbol $\leftarrow$ indicates a *replacement*. Thus, the content of the memory location allocated to a_{ij} is replaced by $a_{ij} - (a_{i1}/a_{11})a_{1j}$, and so on. This is accomplished by the line of pseudocode

$$a_{ij} \leftarrow a_{ij} - (a_{i1}/a_{11})a_{1j}$$

Note that the quantities (a_{i1}/a_{11}) are the *multipliers*. The new coefficient of x_1 in the ith equation will be 0 because $a_{i1} - (a_{i1}/a_{11})a_{11} = 0$.

After the first step, the system will be of the form

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ 0 & a_{23} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & a_{i2} & a_{i3} & \cdots & a_{in} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix}$$

From here on, we will not alter the first equation, nor will we alter any of the coefficients for x_1 (since a multiplier times 0 subtracted from 0 is still 0). Thus, we can mentally ignore the first row and the first column and repeat the process on the smaller system. With the second equation as the pivot equation, we compute for each remaining equation ($3 \leqq i \leqq n$)

$$\begin{cases} a_{ij} \leftarrow a_{ij} - (a_{i2}/a_{22})a_{2j} & (2 \leqq j \leqq n) \\ b_i \leftarrow b_i - (a_{i2}/a_{22})b_2 \end{cases}$$

Just prior to the kth step in the forward elimination, the system will appear as follows:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & & & \cdots & a_{1n} \\ & a_{22} & a_{23} & \cdots & & & \cdots & a_{2n} \\ & & a_{33} & \cdots & & & \cdots & a_{3n} \\ & & & \ddots & & & & \vdots \\ & & & & a_{kk} & \cdots & a_{kn} & \\ & & & & \vdots & & \vdots & \\ & & & & a_{ik} & \cdots & a_{in} & \\ & & & & \vdots & & \vdots & \\ & & & & a_{nk} & \cdots & a_{nn} & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_k \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_k \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix}$$

Here a wedge of 0 coefficients has been created, and the first k equations have been processed and are now fixed. Using the kth equation as the pivot equation, we select multipliers in order to create 0's as coefficients for each x_k below the a_{kk} coefficient. Hence, we compute for each remaining equation $(k+1 \leqq i \leqq n)$

$$\begin{cases} a_{ij} \leftarrow a_{ij} - (a_{ik}/a_{kk})a_{kj} & (k \leqq j \leqq n) \\ b_i \leftarrow b_i - (a_{ik}/a_{kk})b_k \end{cases}$$

Obviously, we must assume that all the divisors in this algorithm are nonzero.

Pseudocode

We now consider the pseudocode for forward elimination. The coefficient array is stored as a double-subscripted array (a_{ij}); the right side of the system of equations is stored as a single-subscripted array (b_i); the solution is computed and stored in a single-subscripted array (x_i). It is easy to see that the following lines of pseudocode carry out the forward elimination phase of naive Gaussian elimination:

```
real array (a_ij)_{n×n}, (b_i)_n
integer i, j, k
  ⋮
for k = 1 to n − 1 do
  for i = k + 1 to n do
    for j = k to n do
      a_ij ← a_ij − (a_ik/a_kk)a_kj
    end do
    b_i ← b_i − (a_ik/a_kk)b_k
  end do
end do
```

Since the multiplier a_{ik}/a_{kk} does not depend on j, it should be moved outside the j loop. Notice also that the new values in column k will be 0, at least theoretically, because when $j = k$, we have

$$a_{ij} \leftarrow a_{ik} - (a_{ik}/a_{kk})a_{kk}$$

Since we expect this to be 0, no purpose is served in computing it. The location where the 0 is being created is a good place to store the multiplier. If these remarks are put into practice, the pseudocode will look like this:

```
real array (a_ij)_{n×n}, (b_i)_n
integer i, j, k
real xmult
  ⋮
for k = 1 to n − 1 do
  for i = k + 1 to n do
    xmult ← a_ik/a_kk
    a_ik ← xmult
    for j = k + 1 to n do
      a_ij ← a_ij − (xmult)a_kj
    end do
    b_i ← b_i − (xmult)b_k
  end do
end do
```

At the beginning of the back substitution phase, the linear system is of the form

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots & \cdots + a_{1n}x_n = b_1 \\ a_{22}x_2 + a_{23}x_3 + \cdots & \cdots + a_{2n}x_n = b_2 \\ a_{33}x_3 + \cdots & + a_{3n}x_n = b_3 \\ \ddots & \vdots \quad \vdots \\ a_{ii}x_i + a_{i,i+1}x_{i+1} + & \cdots + a_{in}x_n = b_i \\ \ddots & \vdots \\ a_{n-1,n-1}x_{n-1} + & a_{n-1,n}x_n = b_{n-1} \\ & a_{nn}x_n = b_n \end{cases}$$

where the a_{ij}'s and b_i's are *not* the original ones from System (6) but instead are the ones that have been altered by the elimination process.

The back substitution starts by solving the nth equation for x_n:

$$x_n = \frac{b_n}{a_{nn}}$$

Then, using the $(n-1)$th equation, we solve for x_{n-1}:

$$x_{n-1} = \frac{b_{n-1} - a_{n-1,n}x_n}{a_{n-1,n-1}}$$

We continue working upward, recovering each x_i by the formula

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j=i+1}^{n} a_{ij}x_j\right) \qquad (i = n-1, n-2, \ldots, 1)$$

Here is pseudocode to do this:

```
real array (a_ij)_{n×n}, (x_i)_n
integer i, j, n
real sum
  ⋮
x_n ← b_n/a_nn
for i = n − 1 to 1 step − 1 do
  sum ← b_i
  for j = i + 1 to n do
    sum ← sum − a_ij x_j
  end do
  x_i ← sum/a_ii
end do
```

Now we put these segments of pseudocode together to form a procedure, called *ngauss*, which is intended to solve a system of n linear equations in n unknowns by the method of naive Gaussian elimination. This pseudocode serves a didactic purpose only; a more robust pseudocode will be developed in the next section.

```
procedure ngauss(n, (a_ij), (b_i), (x_i))
real array (a_ij)_{n×n}, (b_i)_n, (x_i)_n
integer i, j, k, n
real sum, xmult
for k = 1 to n − 1 do
  for i = k + 1 to n do
    xmult ← a_ik/a_kk
    a_ik ← xmult
    for j = k + 1 to n do
      a_ij ← a_ij − (xmult)a_kj
    end do
    b_i ← b_i − (xmult)b_k
  end do
end do
x_n ← b_n/a_nn
for i = n − 1 to 1 step − 1 do
  sum ← b_i
  for j = i + 1 to n do
    sum ← sum − a_ij x_j
```



```
  end do
  x_i ← sum/a_ii
end do
end procedure ngauss
```

Testing the Pseudocode

One good way to test a procedure is to set up an artificial problem whose solution is known beforehand. Sometimes the test problem will include a parameter that can be changed to vary the difficulty. The next example illustrates this.

Fixing a value of n, define the polynomial

$$p(t) = 1 + t + t^2 + \cdots + t^{n-1} = \sum_{j=1}^{n} t^{j-1}$$

The coefficients in this polynomial are all equal to 1. We shall try to recover these known coefficients from n values of the polynomial. We use the values of $p(t)$ at the integers $t = 1 + i$ for $i = 1, 2, \ldots, n$. Denoting the coefficients in the polynomial $x_1, x_2, \ldots, x_n$, we should have

$$\sum_{j=1}^{n}(1+i)^{j-1}x_j = \frac{1}{i}\left[(1+i)^n - 1\right] \qquad (1 \leqq i \leqq n) \tag{7}$$

Here we have used the formula for the sum of a geometric series on the right-hand side; that is,

$$p(1+i) = \sum_{j=1}^{n}(1+i)^{j-1} = \frac{(1+i)^n - 1}{(1+i) - 1} = \frac{1}{i}\left[(1+i)^n - 1\right]$$

Letting $a_{ij} = (1+i)^{j-1}$ and $b_i = [(1+i)^n - 1]/i$ in Equation (7), we have a linear system:

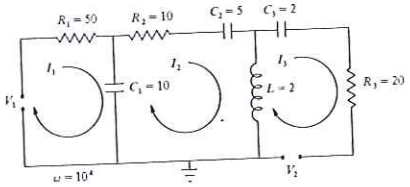
$$\sum_{j=1}^{n} a_{ij}x_j = b_i \qquad (1 \leqq i \leqq n) \tag{8}$$

EXAMPLE 1 Write a pseudocode that solves the system of Equation (8) for various values of n.

Solution Since the naive Gaussian elimination procedure *ngauss* can be used, all that is needed is a calling program. We decide to use $n = 4, 5, 6, 7, 8, 9, 10$ for the test. Here is a suitable pseudocode:

```
program main
real array (a_ij)_{n×n}, (b_i)_n, (x_i)_n
integer i, j, n
for n = 4 to 10 do
```

Use the complex-arithmetic version of *ngauss* and in each case solve the system for the amplitude (in milliamperes) and the phase (in degrees) for each current I_k. *Hint*: When $I_k = \Re(I_k) + i\Im(I_k)$, the amplitude is $|I_k|$, and the phase is $(180^\circ/\pi)\arctan[\Im(I_k)/\Re(I_k)]$. Draw a diagram to show why this is so.



8. Select a reasonable value of n and generate a random $n \times n$ array a using a random-number generator. Define the array b so that the solution of the system

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \qquad (1 \leqq i \leqq n)$$

is $x_j = j$, where $1 \leqq j \leqq n$. Test the naive Gaussian algorithm on this system. *Hint*: You may use the function *random*, which is developed in Chapter 9, to generate the random elements of the (a_{ij}) array.

9. Carry out the test described in the text for procedure *ngauss* but *reverse* the order of the equations. *Hint*: It suffices, in the code, to replace i by $n - i + 1$ in appropriate places.

10. Solve the linear system given in the lead-off example to this chapter using *ngauss*.

6.2 Gaussian Elimination with Scaled Partial Pivoting

Examples Where Naive Gaussian Elimination Fails

To see why the naive Gaussian elimination algorithm is unsatisfactory, consider the system

$$\begin{cases} 0x_1 + x_2 = 1 \\ x_1 + x_2 = 2 \end{cases} \tag{1}$$

The pseudocode constructed in Section 6.1 would attempt to subtract some multiple of the first equation from the second in order to produce a 0 as the coefficient for

x_1 in the second equation. This, of course, is impossible, so the algorithm fails if $a_{11} = 0$.

If a numerical procedure actually fails for some values of the data, then the procedure is probably untrustworthy for values of the data *near* the failing values. To test this dictum, consider the system

$$\begin{cases} \varepsilon x_1 + x_2 = 1 \\ x_1 + x_2 = 2 \end{cases} \tag{2}$$

in which ε is a small number different from 0. Now the naive algorithm of Section 6.1 works and produces first the system

$$\begin{cases} \varepsilon x_1 + x_2 = 1 \\ \left(1 - \dfrac{1}{\varepsilon}\right)x_2 = 2 - \dfrac{1}{\varepsilon} \end{cases} \tag{3}$$

In the back substitution, the arithmetic is as follows:

$$x_2 = \frac{2 - 1/\varepsilon}{1 - 1/\varepsilon} \qquad x_1 = \frac{1 - x_2}{\varepsilon}$$

Now $1/\varepsilon$ will be large, and so if this calculation is performed by a computer that has a fixed word length, then for small values of ε, both $(2 - 1/\varepsilon)$ and $(1 - 1/\varepsilon)$ would be computed as $-1/\varepsilon$.

For example, in an 8-digit decimal machine with a 16-digit accumulator, when $\varepsilon = 10^{-9}$, it follows that $1/\varepsilon = 10^9$. In order to subtract, the computer must interpret the numbers as

$$\frac{1}{\varepsilon} = 10^9 = 0.10000000 \times 10^{10} = 0.1000000000000000 \times 10^{10}$$
$$2 = 0.20000000 \times 10^{1} = 0.0000000002000000 \times 10^{10}$$

Thus, $(1/\varepsilon - 2)$ is computed initially as $0.0999999998000000 \times 10^{10}$ and then rounded to $0.10000000 \times 10^{10} = 1/\varepsilon$.

We conclude that for values of ε sufficiently close to 0, the computer calculates x_2 as 1 and then x_1 as 0. Since the correct solution is

$$x_1 = \frac{1}{1 - \varepsilon} \approx 1 \qquad x_2 = \frac{1 - 2\varepsilon}{1 - \varepsilon} \approx 1$$

the relative error in the computed solution for x_1 is extremely large: 100%.

Actually, the naive Gaussian elimination algorithm works well on examples (1) and (2) if the equations are first permuted:

$$\begin{cases} x_1 + x_2 = 2 \\ x_2 = 1 \end{cases} \quad \text{or} \quad \begin{cases} x_1 + x_2 = 2 \\ \varepsilon x_1 + x_2 = 1 \end{cases}$$

Indeed, the second of these systems becomes

$$\begin{cases} x_1 + x_2 = 2 \\ (1 - \varepsilon)x_2 = 1 - 2\varepsilon \end{cases}$$

after the forward elimination. Then from the back substitution, the solution is computed as

$$x_2 = \frac{1 - 2\varepsilon}{1 - \varepsilon} \approx 1 \qquad x_1 = 2 - x_2 \approx 1$$

The difficulty in System (2) is not due simply to ε being small but rather to its being small relative to other coefficients in the same row. To verify this, consider

$$\begin{cases} x_1 + \dfrac{1}{\varepsilon}x_2 = \dfrac{1}{\varepsilon} \\ x_1 + x_2 = 2 \end{cases} \tag{4}$$

System (4) is mathematically equivalent to (2). The naive Gaussian elimination algorithm fails here, too, because it produces the triangular system

$$\begin{cases} x_1 + \dfrac{1}{\varepsilon}x_2 = \dfrac{1}{\varepsilon} \\ \left(1 - \dfrac{1}{\varepsilon}\right)x_2 = 2 - \dfrac{1}{\varepsilon} \end{cases}$$

and then, in the back substitution, it produces the erroneous result

$$x_2 = \frac{2 - \dfrac{1}{\varepsilon}}{1 - \dfrac{1}{\varepsilon}} \approx 1 \qquad x_1 = \frac{1}{\varepsilon} - \frac{1}{\varepsilon}x_2 \approx 0$$

Gaussian Elimination with Scaled Partial Pivoting

These simple examples should make it clear that the *order* in which we treat the equations significantly affects the accuracy of the elimination algorithm in the computer. In the naive Gaussian elimination algorithm, we use the first equation to eliminate x_1 from the following equations. Then we use the second equation to eliminate x_2 from the following equations, and so on. The order in which the equations are used as pivot equations is the *natural* order $\{1, 2, \ldots, n\}$. Note that the last equation (equation number n) is *not* used as an operating equation with the natural ordering: At no time are multiples of it subtracted from other equations in the naive algorithm.

The Gaussian elimination algorithm now to be described uses the equations in an order that is determined by the actual system being solved. For instance, if the

algorithm were asked to solve System (1) or (2), the order in which the equations would be used as pivot equations would not be the natural order $\{1, 2\}$ but rather $\{2, 1\}$. This order is automatically determined by the computer program. The order in which the equations are employed is denoted by the row vector $[\ell_1, \ell_2, \ldots, \ell_n]$, with ℓ_n not actually being used in the forward elimination phase. Here the ℓ_i are integers from 1 to n in a possibly different order. We call $\boldsymbol{\ell} = [\ell_1, \ell_2, \ldots, \ell_n]$ the **index vector**. The strategy to be described now for determining the index vector is termed **scaled partial pivoting**.

At the beginning, a scale factor must be computed for each equation in the system. Referring to the notation in Section 6.1, we define

$$s_i = \max_{1 \leqq j \leqq n} |a_{ij}| \qquad (1 \leqq i \leqq n)$$

These n numbers are recorded in the **scale vector** $\mathbf{s} = [s_1, s_2, \ldots, s_n]$.

In starting the forward elimination process, we do not arbitrarily use the first equation as the pivot equation. Instead we use the equation for which the ratio $|a_{\ell_1 1}|/s_{\ell_1}$ is greatest. Let ℓ_1 be the index for which this ratio is greatest. Now appropriate multipliers of equation ℓ_1 are subtracted from the other equations in order to create 0's as coefficients for each x_1 except in the pivot equation.

The best way of keeping track of the indices is as follows: At the beginning, define the index vector $\boldsymbol{\ell}$ to be $[\ell_1, \ell_2, \ldots, \ell_n] = [1, 2, \ldots, n]$. Select j to be the index associated with the largest ratio in the set

$$\left\{ \frac{|a_{\ell_i 1}|}{s_{\ell_i}} : 1 \leqq i \leqq n \right\}$$

Now interchange ℓ_j with ℓ_1 in the index vector $\boldsymbol{\ell}$. Next, use multipliers

$$\frac{a_{\ell_i 1}}{a_{\ell_1 1}}$$

times row ℓ_1 and subtract from equations ℓ_i for $2 \leqq i \leqq n$. It is important to note that only entries in $\boldsymbol{\ell}$ are being interchanged and *not* the equations. This eliminates the time-consuming and unnecessary process of moving the coefficients of equations around in the computer memory!

In the second step, the ratios

$$\left\{ \frac{|a_{\ell_i 2}|}{s_{\ell_i}} : 2 \leqq i \leqq n \right\}$$

are scanned. With j the index for the largest ratio, interchange ℓ_j with ℓ_2 in $\boldsymbol{\ell}$. Then multipliers

$$\frac{a_{\ell_i 2}}{a_{\ell_2 2}}$$

times equation ℓ_2 are subtracted from equations ℓ_i for $3 \leqq i \leqq n$.

At step k, select j to be the index corresponding to the largest of the ratios

$$\left\{\frac{|a_{\ell_i k}|}{s_{\ell_i}} : k \leqq i \leqq n\right\}$$

and interchange ℓ_j and ℓ_k in index vector ℓ. Then multipliers

$$\frac{a_{\ell_i k}}{a_{\ell_k k}}$$

times pivot equation ℓ_k are subtracted from equations ℓ_i for $k+1 \leqq i \leqq n$.

Notice that the scale factors are *not* changed after each pivot step. Intuitively, one might feel that after each pivot and elimination, the remaining (modified) coefficients should be used to recompute the scale factors instead of using the original scale vector. Of course, this could be done, but it is generally believed that the extra computations involved in this procedure are not worthwhile in the majority of linear systems. The reader is encouraged to explore this question. (See Computer Problem **16.**)

Numerical Example

We are not quite ready to write pseudocode, but let us consider what has been outlined in a concrete example. Consider this system:

$$\begin{bmatrix} 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \\ 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -19 \\ -34 \\ 16 \\ 26 \end{bmatrix}$$

The index vector is $\ell = [1,2,3,4]$ at the beginning. The scale vector does not change throughout the procedure and is $s = [13, 18, 6, 12]$. To determine the first pivot row, we look at four ratios:

$$\left\{\frac{|a_{\ell_i 1}|}{s_{\ell_i}} : i = 1,2,3,4\right\} = \left\{\frac{3}{13}, \frac{6}{18}, \frac{6}{6}, \frac{12}{12}\right\}$$

We select the index j as the *first* occurrence of the largest value of these ratios. In this example, the largest of these occurs for the index $j = 3$. So row 3 is to be the pivot equation in step 1 ($k = 1$) of the elimination process. In the index vector ℓ, entries ℓ_k and ℓ_j are interchanged so that the new index vector is $\ell = [3,2,1,4]$. Thus, the pivot equation is ℓ_k, which is $\ell_1 = 3$. Now appropriate multiples of the third equation are subtracted from the other equations so as to create 0's as coefficients for x_1 in each of those equations. Explicitly, $\frac{1}{2}$ times row 3 is subtracted from row 1, -1 times row 3 is subtracted from row 2, and 2 times row 3 is subtracted from row 4. The result is

$$\begin{bmatrix} 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \\ 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -27 \\ -18 \\ 16 \\ -6 \end{bmatrix}$$

In the next step ($k = 2$), we use the index vector $\ell = [3,2,1,4]$ and scan the ratios

$$\left\{\frac{|a_{\ell_i 2}|}{s_{\ell_i}} : i = 2,3,4\right\} = \left\{\frac{2}{18}, \frac{12}{13}, \frac{4}{12}\right\}$$

looking for the largest value. Hence, we set $j = 3$ and interchange ℓ_k with ℓ_j in the index vector. Thus, the index vector becomes $\ell = [3,1,2,4]$. The pivot equation for step 2 in the elimination is now $\ell_2 = 1$. Next, multiples of the first equation are subtracted from the second equation and the fourth equation. The appropriate multiples are $-\frac{1}{6}$ and $\frac{1}{3}$, respectively. The result is

$$\begin{bmatrix} 0 & -12 & 8 & 1 \\ 0 & 0 & \frac{13}{3} & -\frac{83}{6} \\ 6 & -2 & 2 & 4 \\ 0 & 0 & -\frac{2}{3} & \frac{5}{3} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -27 \\ -\frac{45}{2} \\ 16 \\ 3 \end{bmatrix}$$

The third and final step ($k = 3$) is to examine the ratios

$$\left\{\frac{|a_{\ell_i 3}|}{s_{\ell_i}} : i = 3,4\right\} = \left\{\frac{13/3}{18}, \frac{2/3}{12}\right\}$$

with the index vector $\ell = [3,1,2,4]$. The larger value is the first, so we set $j = 3$. Since this is step 3, interchanging ℓ_k with ℓ_j leaves the index vector as before, $\ell = [3,1,2,4]$. The pivot equation is $\ell_3 = 2$, and we subtract $-\frac{2}{13}$ times the second equation from the fourth equation. So the forward elimination phase ends with the final system

$$\begin{bmatrix} 0 & -12 & 8 & 1 \\ 0 & 0 & \frac{13}{3} & -\frac{83}{6} \\ 6 & -2 & 2 & 4 \\ 0 & 0 & 0 & -\frac{6}{13} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -27 \\ -\frac{45}{2} \\ 16 \\ -\frac{6}{13} \end{bmatrix}$$

The order in which the pivot equations were selected is displayed in the final index vector $\ell = [3,1,2,4]$.

Now, reading the entries in the index vector from the last to the first, we have the order in which the back substitution is to be performed. The solution is obtained by using equation $\ell_4 = 4$ to determine x_4, and then equation $\ell_3 = 2$ to find x_3, and so on. Clearly, we have

$$x_4 = \frac{1}{-6/13}[-6/13] = 1$$

$$x_3 = \frac{1}{13/3}[(-45/2) + (83/6)(1)] = -2$$

$$x_2 = \frac{1}{-12}[-27 - 8(-2) - 1(1)] = 1$$

$$x_1 = \frac{1}{6}[16 + 2(1) - 2(-2) - 4(1)] = 3$$

Hence, the solution is

$$x = \begin{bmatrix} 3 \\ 1 \\ -2 \\ 1 \end{bmatrix}$$

Pseudocode

The algorithm as it will be programmed carries out the forward elimination phase on the coefficient array (a_{ij}) only. The right-hand-side array (b_i) is treated in the next phase. This method is adopted because it is more efficient if several systems must be solved with the same array (a_{ij}) but differing (b_i)'s. Because we wish to treat (b_i) later, it is necessary to store not only the index array but also the various multipliers that are used. These multipliers are conveniently stored in array (a_{ij}) in the positions where the 0 entries would have been created.

We are now ready to write a procedure for forward elimination with scaled partial pivoting. Our approach is to modify procedure *ngauss* of Section 6.1 by introducing scaling and indexing arrays. The procedure that carries out Gaussian elimination with scaled partial pivoting on the square array (a_{ij}) is called *gauss*. Its calling sequence is $(n, (a_{ij}), (\ell_i))$, where (a_{ij}) is the $n \times n$ coefficient array and (ℓ_i) is the index array for ℓ. In the pseudocode, (s_i) is the scale array.

```
procedure gauss(n, (a_ij), (ℓ_i))
real array (a_ij)_{n×n}, (ℓ_i)_n
real array allocate (s_i)_n
integer i, j, k, n
real r, rmax, smax, xmult
for i = 1 to n do
  ℓ_i ← i
  smax ← 0
  for j = 1 to n do
    smax ← max(smax, a_ij)
  end do
  s_i ← smax
end do
```



```
for k = 1 to n − 1 do
  rmax ← 0
  for i = k to n do
    r ← |a_{ℓ_i k}/s_{ℓ_i}|
    if (r > rmax) then
      rmax ← r
      j ← i
    end if
  end do
  ℓ_j ↔ ℓ_k
  for i = k + 1 to n do
    xmult ← a_{ℓ_i k}/a_{ℓ_k k}
    a_{ℓ_i k} ← xmult
    for j = k + 1 to n do
      a_{ℓ_i j} ← a_{ℓ_i j} − (xmult)a_{ℓ_k j}
    end do
  end do
end do
end procedure gauss
```

A detailed explanation of the above procedure is now presented. In the first loop, the initial form of the index array is being established—namely, $\ell_i = i$. Then the scale array (s_i) is computed.

The statement **for** $k = 1$ **to** $n - 1$ **do** initiates the principal outer loop. The index k is the subscript of the variable whose coefficients will be made 0 in the array (a_{ij}); that is, k is the index of the column in which new 0's are to be created. Remember that the 0's in the array (a_{ij}) do not actually appear because those storage locations are used for the multipliers. This fact can be seen in the line of the procedure where *xmult* is stored in the array (a_{ij}).

Once k has been set, the first task is to select the correct pivot row, which is done by computing $|a_{\ell_i k}|/s_{\ell_i}$ for $i = k, k+1, \ldots, n$. The next set of lines in the pseudocode are calculating this greatest ratio, called *rmax* in the routine, and the index j where it occurs. Next, ℓ_k and ℓ_j are interchanged in the array (ℓ_i).

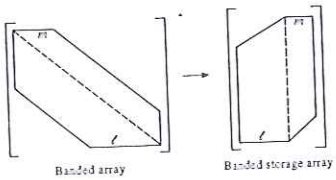
The arithmetic modifications in the array (a_{ij}) due to subtracting multiples of row ℓ_k from rows $\ell_{k+1}, \ell_{k+2}, \ldots, \ell_n$ all occur in the final lines. First the multiplier is computed and stored; then the subtraction occurs in a loop.

Caution: Values in array (a_{ij}) that result as *output* from procedure *gauss* are not the same as those in array (a_{ij}) at *input*. If the original array must be retained, one should store a duplicate of it in another array.

In the procedure *ngauss* for naive Gaussian elimination from Section 6.1, the right-hand side $\boldsymbol{b}$ was modified during the forward elimination phase; however, this was not done in the procedure *gauss*. Therefore, we need to update $\boldsymbol{b}$ before considering the back substitution phase. For simplicity, we discuss updating $\boldsymbol{b}$ for the naive forward elimination first. Stripping out the pseudocode from *ngauss* that involves the (b_i) array in the forward elimination phase, we obtain

in the figure, then the $n \times (\ell + m + 1)$ array in banded storage mode would be as shown. The main diagonal would be the $\ell + 1$st column of the new array. Write and test a procedure for solving a linear system with the coefficient matrix stored in banded storage mode.

20. An $n \times n$ symmetric banded coefficient matrix with m subdiagonals and m superdiagonals can be stored in **symmetric banded storage** mode in an $n \times (m + 1)$ array. Only the main diagonal and subdiagonals are stored so that the main diagonal is the last column in the new array, shown in the figure. Write and test a procedure for solving a linear system with the coefficient matrix stored in symmetric banded storage mode.



Banded array Banded storage array

6.4 *LU* Factorization

This section requires some basic knowledge of linear algebra. Some of the notation and concepts of linear algebra are presented in Appendix A. This material can be skipped without loss of continuity.

An $n \times n$ system of linear equations can be written in matrix form

$$Ax = b \tag{1}$$

where the coefficient matrix A has the form

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix}$$

Our main objective is to show that the naive Gaussian algorithm applied to A yields a factorization of A into a product of two simple matrices, one *unit lower triangular*:

$$L = \begin{bmatrix} 1 & & & & \\ \ell_{21} & 1 & & & \\ \ell_{31} & \ell_{32} & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & 1 \end{bmatrix}$$

and the other upper triangular:

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ & u_{22} & u_{23} & \cdots & u_{2n} \\ & & u_{33} & \cdots & u_{3n} \\ & & & \ddots & \vdots \\ & & & & u_{nn} \end{bmatrix}$$

In short, we refer to this as an ***LU* factorization** of A; that is,

$$A = LU$$

Numerical Example

The system of Equations (2) of Section 6.1 can be written succinctly in matrix form:

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ 26 \\ -19 \\ -34 \end{bmatrix} \tag{2}$$

Furthermore, the operations that led from this system to Equation (5) of Section 6.1—that is, the system

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ -6 \\ -9 \\ -3 \end{bmatrix} \tag{3}$$

could be effected by an appropriate matrix multiplication. The forward elimination phase can be interpreted as starting from (1) and proceeding to

$$MAx = Mb \tag{4}$$

where M is a matrix chosen so that MA is the coefficient matrix for the System (3). Hence, we have

$$MA = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix} \equiv U$$

which is an upper triangular matrix.

The first step of naive Gaussian elimination results in Equation (3) of Section 6.1 or the system

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ -6 \\ -27 \\ -18 \end{bmatrix}$$

This step can be accomplished by multiplying (1) by a lower triangular matrix M_1:

$$M_1Ax = M_1b$$

where

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -\frac{1}{2} & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Notice the special form of M_1. The diagonal elements are all 1's, and the only other nonzero elements are in the first column. These numbers are the *negatives of the multipliers* located in the positions where they created 0's as coefficients in step 1 of the forward elimination phase. To continue, step 2 resulted in Equation (4) of Section 6.1 or the system

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 4 & -13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ -6 \\ -9 \\ -21 \end{bmatrix}$$

which is equivalent to

$$M_2M_1Ax = M_2M_1b$$

where

$$M_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -3 & 1 & 0 \\ 0 & \frac{1}{2} & 0 & 1 \end{bmatrix}$$

Again, M_2 differs from an identity matrix by the presence of the negatives of the multipliers in the second column from the diagonal down. Finally, step 3 gives System (3), which is equivalent to

$$M_3M_2M_1Ax = M_3M_2M_1b$$

where

$$M_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -2 & 1 \end{bmatrix}$$

Now the forward elimination phase is complete, and with

$$M = M_3M_2M_1 \tag{5}$$

we have the upper triangular coefficient System (3).

Using Equations (4) and (5), we can give a different interpretation of the forward elimination phase of naive Gaussian elimination. Now we see that

$$\begin{aligned} A &= M^{-1}U \\ &= M_1^{-1}M_2^{-1}M_3^{-1}U \\ &= LU \end{aligned}$$

Since each M_k has such a special form, its inverse is obtained by simply changing the signs of the negative multiplier entries! Hence, we have

$$\begin{aligned} L &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & -\frac{1}{2} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ \frac{1}{2} & 3 & 1 & 0 \\ -1 & -\frac{1}{2} & 2 & 1 \end{bmatrix} \end{aligned}$$

It is somewhat amazing that L is a unit lower triangular matrix composed of the multipliers. Notice that in forming L, we did not determine M first and then compute $M^{-1} = L$. (Why?)

It is easy to verify that

$$LU = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ \frac{1}{2} & 3 & 1 & 0 \\ -1 & -\frac{1}{2} & 2 & 1 \end{bmatrix} \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix} = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} = A$$

We see that A is **factored** or **decomposed** into a unit lower triangular matrix L and an upper triangular matrix U. The matrix L consists of the multipliers located in the positions of the elements they annihilated from A, of unit diagonal elements, and of 0 upper triangular elements. In fact, we now know the general form of L and can just write it down directly using the multipliers *without* forming the M_k's and the M_k^{-1}'s. The matrix U is upper triangular (not necessarily unit diagonal) and is the final coefficient matrix after the forward elimination phase is completed.

It should be noted that the pseudocode *ngauss* of Section 6.1 replaces the original coefficient matrix with its LU factorization. The elements of U are in the upper triangular part of the (a_{ij}) array including the diagonal. The entries below the main diagonal in L (that is, the multipliers) are found below the main diagonal in the (a_{ij}) array. Since it is known that L has a unit diagonal, nothing is lost by not storing the 1's. [In fact, we have run out of room in the (a_{ij}) array anyway!]

Formal Derivation

In order to see formally how the Gaussian elimination (in naive form) leads to an LU factorization, it is necessary to show that each row operation used in the algorithm can be effected by multiplying A on the left by an elementary matrix. Specifically, if we wish to subtract λ times row p from row q, we first apply this operation to the $n \times n$ identity matrix in order to create an elementary matrix M_{qp}. Then we form the matrix product $M_{qp}A$.

Before proceeding, let us verify that $M_{qp}A$ is obtained by subtracting λ times row p from row q in matrix A. Assume that $p < q$ (for in the naive algorithm this is always true). Then the elements of $M_{qp} = (m_{ij})$ are

$$m_{ij} = \begin{cases} 1 & \text{if } i = j \\ -\lambda & \text{if } i = q \text{ and } j = p \\ 0 & \text{in all other cases} \end{cases}$$

Therefore, the elements of $M_{qp}A$ are given by

$$(M_{pq}A)_{ij} = \sum_{s=1}^{n} m_{is}a_{sj} = \begin{cases} a_{ij} & \text{if } i \neq q \\ a_{qj} - \lambda a_{pj} & \text{if } i = q \end{cases}$$

The qth row of $M_{qp}A$ is the sum of the qth row of A and $-\lambda$ times the pth row of A, as was to be proved.

The kth step of Gaussian elimination corresponds to the matrix M_k, which is the product of $n - k$ elementary matrices:

$$M_k = M_{nk}M_{n-1,k}\cdots M_{k+1,k}$$

Notice that each elementary matrix M_{ik} here is lower triangular because $i > k$ and therefore M_k is also lower triangular. If we carry out the Gaussian forward elimination process on A, the result will be an upper triangular matrix U. On the other hand, the result is obtained by applying a succession of factors like M_k to the

left of A. Hence, the entire process is summarized by writing

$$M_{n-1}\cdots M_2M_1A = U$$

Since each M_k is invertible, we have

$$A = M_1^{-1}M_2^{-1}\cdots M_{n-1}^{-1}U$$

Each M_k is lower triangular with 1's on its main diagonal (unit lower triangular). Each inverse M_k^{-1} has the same property, and the same is true of their product. Hence, the matrix

$$L = M_1^{-1}M_2^{-1}\cdots M_{n-1}^{-1} \tag{6}$$

is unit lower triangular, and we have

$$A = LU$$

This is the so-called LU factorization of A. Our construction of it depends upon *not* encountering any 0 divisors in the algorithm. It is easy to give examples of matrices that have no LU factorization; one of the simplest is $A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$. (See Problem 4.)

LU FACTORIZATION THEOREM

Let $A = (a_{ij})$ be an $n \times n$ matrix. Assume that the forward elimination phase of the naive Gaussian algorithm is applied to A without encountering any 0 divisors. Let the resulting matrix be denoted by $\tilde{A} = (\tilde{a}_{ij})$. If

$$L = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \tilde{a}_{21} & 1 & 0 & \cdots & 0 \\ \tilde{a}_{31} & \tilde{a}_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \tilde{a}_{n1} & \tilde{a}_{n2} & \cdots & \tilde{a}_{n,n-1} & 1 \end{bmatrix}$$

and

$$U = \begin{bmatrix} \tilde{a}_{11} & \tilde{a}_{12} & \tilde{a}_{13} & \cdots & \tilde{a}_{1n} \\ 0 & \tilde{a}_{22} & \tilde{a}_{23} & \cdots & \tilde{a}_{2n} \\ 0 & 0 & \tilde{a}_{33} & \cdots & \tilde{a}_{3n} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \tilde{a}_{nn} \end{bmatrix}$$

then $A = LU$.

Proof We define the Gaussian algorithm formally as follows. Let $A^{(1)} = A$. Then we compute $A^{(2)}, A^{(3)}, \ldots, A^{(n)}$ recursively by the naive Gaussian algorithm, following these equations:

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} \qquad (\text{if } i \leqq k \text{ or } j < k) \tag{7}$$

$$a_{ij}^{(k+1)} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \qquad (\text{if } i > k \text{ and } j = k) \tag{8}$$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \left(\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}\right)a_{kj}^{(k)} \qquad (\text{if } i > k \text{ and } j > k) \tag{9}$$

These equations describe in a precise form the forward elimination phase of the naive Gaussian elimination algorithm. For example, Equation (7) states that in proceeding from $A^{(k)}$ to $A^{(k+1)}$, we do not alter rows $1, 2, \ldots, k$ or columns $1, 2, \ldots, k-1$. Equation (8) shows how the multipliers are computed and stored in passing from $A^{(k)}$ to $A^{(k+1)}$. Finally, Equation (9) shows how multiples of row k are subtracted from rows $k+1, k+2, \ldots, n$ in order to produce $A^{(k+1)}$ from $A^{(k)}$.

Notice that $A^{(n)}$ is the final result of the process. (It was referred to as $\tilde{A}$ in the statement of the theorem.) The formal definitions of $L = (\ell_{ik})$ and $U = (u_{ij})$ are, therefore,

$$\ell_{ik} = 1 \qquad (i = k) \tag{10}$$

$$\ell_{ik} = a_{ik}^{(k)} \qquad (k < i) \tag{11}$$

$$\ell_{ik} = 0 \qquad (k > i) \tag{12}$$

$$u_{ij} = a_{ij}^{(k)} \qquad (j \geqq k) \tag{13}$$

$$u_{ij} = 0 \qquad (j < k) \tag{14}$$

Now we draw some consequences of these equations. First, it follows immediately from Equation (7) that

$$a_{ij}^{(i)} = a_{ij}^{(i+1)} = \cdots = a_{ij}^{(n)} \tag{15}$$

Likewise, we have from Equation (7)

$$a_{ij}^{(j+1)} = a_{ij}^{(j+2)} = \cdots = a_{ij}^{(n)} \qquad (j < n) \tag{16}$$

From Equations (16) and (8), we have now

$$a_{ij}^{(n)} = a_{ij}^{(j+1)} = \frac{a_{ij}^{(j)}}{a_{jj}^{(j)}} \qquad (j < n) \tag{17}$$

From Equations (17) and (11), it follows that

$$\ell_{ik} = a_{ik}^{(n)} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \qquad (k < i) \tag{18}$$

From Equations (13) and (15), we have

$$u_{ij} = a_{ij}^{(n)} = a_{ij}^{(i)} \qquad (k \leqq j) \tag{19}$$

With the aid of all these equations, we can now prove that $LU = A$. First, consider the case $i \leqq j$. Then

$$\begin{aligned} (LU)_{ij} &= \sum_{k=1}^{n} \ell_{ik}u_{kj} && \text{[definition of multiplication]} \\ &= \sum_{k=1}^{i} \ell_{ik}u_{kj} && \text{[by Equation (12)]} \\ &= \sum_{k=1}^{i-1} \ell_{ik}u_{kj} + u_{ij} && \text{[by Equation (10)]} \\ &= \sum_{k=1}^{i-1} \left[\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}\right]a_{kj}^{(k)} + a_{ij}^{(i)} && \text{[by Equations (18) and (19)]} \\ &= \sum_{k=1}^{i-1} \left[a_{ij}^{(k)} - a_{ij}^{(k+1)}\right] + a_{ij}^{(i)} && \text{[by Equation (9)]} \\ &= a_{ij}^{(1)} = a_{ij} \end{aligned}$$

In the remaining case, $i > j$, we have

$$\begin{aligned} (LU)_{ij} &= \sum_{k=1}^{n} \ell_{ik}u_{kj} && \text{[definition of multiplication]} \\ &= \sum_{k=1}^{j} \ell_{ik}u_{kj} && \text{[by Equation (14)]} \\ &= \sum_{k=1}^{j} \left[\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}\right]a_{kj}^{(k)} && \text{[by Equations (18) and (19)]} \\ &= \sum_{k=1}^{j-1} \left[\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}\right]a_{kj}^{(k)} + a_{ij}^{(j)} \end{aligned}$$

$$= \sum_{k=1}^{j-1} \left[a_{ij}^{(k)} - a_{ij}^{(k+1)} \right] + a_{ij}^{(j)} \qquad \text{[by Equation (9)]}$$

$$= a_{ij}^{(1)} = a_{ij} \qquad \blacksquare$$

Solving Linear Systems Using *LU* Factorization

Once the LU factorization of A is available, we can solve the system

$$Ax = b$$

by writing

$$LUx = b$$

Then we solve two triangular systems:

$$Lz = b \tag{20}$$

for z and

$$Ux = z \tag{21}$$

for x. This is particularly useful for problems that involve the same coefficient matrix A and many different right-hand vectors b.

Since L is unit lower triangular, z is obtained by the pseudocode

```
real array (b_i)_n, (ℓ_ij)_{n×n}, (z_i)_n
integer i, n
  ⋮
z_1 ← b_1
for i = 2 to n do
   z_i ← b_i − Σ_{j=1}^{i−1} ℓ_ij z_j
end do
```

Likewise, x is obtained by the pseudocode

```
real array (u_ij)_{n×n}, (x_i)_n, (z_i)_n
integer i, n
  ⋮
x_n ← z_n / u_nn
for i = n − 1 to 1 step − 1 do
   x_i ← ( z_i − Σ_{j=i+1}^{n} u_ij x_j ) / u_ii
end do
```


The first of these two algorithms applies the forward phase of Gaussian elimination to the right-hand-side vector b. [Recall that the ℓ_{ij}'s are the *multipliers* that have been stored in the array (a_{ij}).] The easiest way to verify this assertion is to use Equation (6) and to rewrite the equation

$$Lz = b$$

in the form

$$M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1} z = b$$

From this we get immediately

$$z = M_{n-1} \cdots M_2 M_1 b$$

Thus, the same operations used to reduce A to U are to be used on b in order to produce z.

Another way to solve Equation (20) is to note that what must be done is to form

$$M_{n-1} M_{n-2} \cdots M_2 M_1 b$$

This can be accomplished using only the array (b_i) by putting the results back into b; that is,

$$b \leftarrow M_k b$$

We know what M_k looks like because it is made up of negative multipliers that have been saved in the array (a_{ij}). Clearly, we have

$$M_k b = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & -a_{k+1,k} & 1 & & & \\ & & \vdots & & \ddots & & \\ & & -a_{ik} & & & 1 & \\ & & \vdots & & & & \ddots \\ & & -a_{nk} & & & & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ b_k \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix}$$

The entries b_1 to b_k are not changed by this multiplication, while b_i (for $i \geq k+1$) is replaced by $-a_{ik} b_k + b_i$. Hence, the following pseudocode updates the array (b_i) based on the stored multipliers in the array a:

```
real array (a_ij)_{n×n}, (b_i)_n
integer i, k, n
```



```
  ⋮
for k = 1 to n − 1 do
   for i = k + 1 to n do
      b_i ← b_i − a_ik b_k
   end do
end do
```

This pseudocode should be familiar. It is the process for updating b from Section 6.2.

The algorithm for solving Equation (21) is the back substitution phase of the naive Gaussian elimination process.

Computing A^{-1}

In some applications, such as in statistics, it may be necessary to compute the inverse of a matrix A and explicitly display it as A^{-1}. This can be done by using procedures *gauss* and *solve*. If an $n \times n$ matrix A has an inverse, it is an $n \times n$ matrix X with the property that

$$AX = I \tag{22}$$

where I is the identity matrix. If $X^{(j)}$ denotes the jth column of X and $I^{(j)}$ denotes the jth column of I, then matrix Equation (22) can be written as

$$A[X^{(1)}, X^{(2)}, \ldots, X^{(n)}] = [I^{(1)}, I^{(2)}, \ldots, I^{(n)}]$$

This can be written as n linear systems of equations of the form

$$AX^{(j)} = I^{(j)} \qquad (1 \leq j \leq n)$$

Now use procedure *gauss* once to produce a factorization of A and use procedure *solve* n times with the vectors $I^{(j)}$ $(1 \leq j \leq n)$. This is equivalent to solving, one at a time, for the columns of A^{-1}, which are $X^{(j)}$. Hence,

$$A^{-1} = [X^{(1)}, X^{(2)}, \ldots, X^{(n)}]$$

A word of caution on computing the inverse of a matrix: When solving a linear system $Ax = b$, it is not advisable to determine A^{-1} and then compute $x = A^{-1}b$ because this requires many unnecessary calculations, compared to directly solving $Ax = b$ for x.

Example of a Software Package

The MATLAB input is as follows.


```
A = [ 6,  -2,  2,   4,
     12,  -8,  6,  10,
      3, -13,  9,   3,
     -6,   4,  1, -18]

[L, U, P] = lu(A)
```

The resulting MATLAB output is as follows.

```
L =
    1.0000         0         0         0
    0.2500    1.0000         0         0
   -0.5000         0    1.0000         0
    0.5000   -0.1818    0.0909    1.0000

U =
   12.0000   -8.0000    6.0000   10.0000
         0  -11.0000    7.5000    0.5000
         0         0    4.0000  -13.0000
         0         0         0    0.2727

P =
     0     1     0     0
     0     0     1     0
     0     0     0     1
     1     0     0     0
```

This corresponds to the factorization

$$PA = LU$$

where P is a permutation matrix corresponding to the pivoting strategy used.

PROBLEMS 6.4

1. Using naive Gaussian elimination, factor the following matrices so that $A = LU$, where L is a unit lower triangular matrix and U is an upper triangular matrix.

   a. $A = \begin{bmatrix} 3 & 0 & 3 \\ 0 & -1 & 3 \\ 1 & 3 & 0 \end{bmatrix}$

   b. $A = \begin{bmatrix} 1 & 0 & \frac{1}{3} & 0 \\ 0 & 1 & 3 & -1 \\ 3 & -3 & 0 & 6 \\ 0 & 2 & 4 & -6 \end{bmatrix}$